3.1 栈

栈的基本概念

只允许在一端进行插入或删除操作的线性表　　先进后出

结构
- 栈顶（Top）：线性表允许进行插入删除的那一端
- 栈底（Bottom）：固定的，不允许进行插入和删除的另一端

卡特兰数　n个不同元素进栈,出栈元素不同排列的个数为 $\frac{1}{n+1}C_{2n}^{n}$

栈的基本操作
- InitStack (&S):初始化一个空栈S
- StackEmpty (S):判断一个栈是否为空，若栈S为空则返回true.否则返回falseo Push (&S, x):进栈,若栈S未满,则将x加入使之成为新栈顶
- Pop (&S， &x):出栈,若栈S非空,则弹出栈顶元素，并用x返回
- GetTop(S，&x)：读栈顶元素，若栈S非空，则用x返回栈顶元素
- DestroyStack (&S):销毁栈，并释放栈S占用的存储空间("&"表示引用调用)

在解答算法题时，若题干未做出限制，则可直接使用这些基本的操作函数

栈的顺序存储结构

顺序栈的实现

采用顺序存储的栈称为顺序栈，它利用一组地址连续的存储单元存放自栈底到栈顶的数据元素，同时附设一个指针(top)指示当前栈顶元素的位置

基本结构与操作
- 栈顶指针:S.top,初始时设置S.top=-1
- 栈顶元素:S.data[S.top]
- 进栈操作:栈不满时,栈顶指针先加1,再送值到栈顶元素
- 出栈操作:栈非空时,先取栈顶元素值,再将栈顶指针减1
- 栈空条件:S.top==-1
- 栈满条件:S.top==MaxSize-1;栈长:S.top+1

顺序栈的基本操作

初始化
```
void InitStack(SqStack &S){
   S.top=-1;                    //初始化栈顶指针
}
```

判栈空
```
bool StackEmpty(SqStack S){
   if(S.top==-1)                //栈空
      return true;
   else                         //不空
      return false;
}
```

进栈
```
bool Push(SqStack &S,ElemType x){
  if(S.top==MaxSize-1)          //栈满，报错
    return false;
  S.data[++S.top]=x;            //指针先加 1，再入栈
  return true;
}
```

出栈
```
bool Pop(SqStack &S,ElemType &x){
   if(S.top==-1)                //栈空，报错
      return false;
   x=S.data[S.top--];           //先出栈，指针再减 1
   return true;
}
```

读栈顶元素
```
bool GetTop(SqStack S,ElemType &x){
   if(S.top==-1)                //栈空，报错
      return false;
   x=S.data[S.top];             //x 记录栈顶元素
   return true;
}
```

共享栈

让两个顺序栈共享一个一维数组空间，将两个栈的栈底分别设置在共享空间的两端，两个栈顶向共享空间的中间延伸

基本原则
- 两个栈的栈顶指针都指向栈顶元素,top0=-1时0号栈为空,top1=MaxSize时1号栈为空
- 当两个栈顶指针相邻（topl-top0=1）时,判断为栈满
- 当0号栈进栈时top0先加1 再赋值,1号栈进栈时 top1先减1再赋值;出栈时则刚好相反

存取数据的时间复杂度均为O(1)

栈的链式存储结构

采用单链表实现，并规定所有操作都是在单链表的表头进行

优点
- 便于多个栈共享存储空间和提高其效率
- 且不存在栈满上溢的情况

3.2队列（上）

- **队列的基本概念**
  - 队列的定义
    - 是一种操作受限的线性表，只允许在表的一端进行插入，而在表 的另一端进行删除
    - 向队列中插入元素称为入队或进 队；删除元素称为出队或离队
    - 先进先出
    - 结构
      - 队头(Front）：允许删除的一端，又称队首
      - 队尾(Rear)：允许插入的一端
      - 空队列：不含任何元素的空表
  - 队列基本操作
    - InitQueue (&Q):初始化队列，构造一个空队列Q
    - QueueEmpty (Q):判队列空，若队列Q为空返回true.否则返回false
    - EnQueue (&Q，x):入队，若队列Q未满，将x加入，使之成为新的队尾
    - DeQueue (&Q，&x)：出队，若队列Q非空，删除队头元素，并用x返回
    - GetHead(Q，&x)：读队头元素，若队列Q非空，则将队头元素赋值给X
- **队列的顺序存储结构**
  - 队列的顺序存储
    - 分配一块连续的存储单元存放队列中的元素，并附设两个指针
    - 队头指针front指向队头元素，队尾指针rear指向队尾元素的下一个位置（具体问题具体分析）
    - 基本操作
      - 初始状态（队空条件）:Q.front==Q.rear==0
      - 进队操作:队不满时,先送值到队尾元素,再将队尾指针加1
      - 出队操作:队不空时,先取队头元素值,再将队头指针加1
    - 假溢出
      - 这种溢出并不是真正的溢出,在data 数组中依然存在可以存放元素的空位置
  - 循环队列
    - 把存储队列元素的表从逻辑上视为一个环，称为循环队列
    - 基本操作
      - 初始时:Q.front=Q.rear=0
      - 队首指针进1:Q.front=（Q.front+1）%MaxSize
      - 队尾指针进1:Q.rear=(Q.rear+1)%MaxSize
      - 队列长度:(Q.rear+MaxSize-Q.front)%MaxSize。
    - 判断条件
      - 队空：Q.front==Q.rear
      - 牺牲一个单元来区分队空和队满,即"队头指针在队尾指针的下一位置作为队满的标志"
        - 队满条件:(Q.rear+1)%MaxSize==Q.front
        - 队空条件:Q.front==Q.rear
        - 队列中元素的个数:（Q.rear-Q.front+MaxSize）%MaxSize
      - 队满
        - 类型中增设表示元素个数的数据成员。
          - 队空：为Q.size==0
          - 队满：Q.size==MaxSize
        - 类型中增设 tag 数据成员,以区分是队满还是队空
          - tag 等于0时,若因删除导致Q.front==Q.rear,则为队空
          - tag等于1时,若因插入导致Q.front==Q.rear,则为队满

3.2队列（下）

- 队列的链式存储结构
  - 队列的链式存储
    - 队列的链式表示称为链队列，是一个同时带有队头指针和队尾指针的单链表。
    - 头指 针指向队头结点
    - 尾指针指向队尾结点
    - 当Q.front==NULL且Q.rear==NULL时,链式队列为空
  - 优点
    - 适合于数据元素变动比较大的情形
    - 不存在队列满且产生溢出的问题
- 双端队列
  - 概述
    - 双端队列是指允许两端都可以进行入队和出队操作的队列
    - 逻辑结构仍是线性结构
    - 将队列的两端分别称为前端和后端，两端都可以入队和出队
  - 分类
    - 输出受限的双端队列：允许在一端进行插入和删除，但在另一端只允许插入的双端队列
    - 输入受限的双端队列：允许在一端进行插入和删除，但在另一端只允许删除的双端队列

3.3~4矩阵的压缩存储（上）

- **栈的应用**
  - 括号匹配
  - 表达式求值
    - 中缀表达式转后缀表达式（手算）
      - ①确定中缀表达式中各个运算符的运算顺序
      - ② 选择下一个运算符，按照「左操作数 右操作数 运算符」的方式组合成一个新的操作数
      - ③ 如果还有运算符没被处理，就继续 ②
    - 后缀表达式的计算
      - 手算：从左往右扫描，每遇到一个运算符，就让运算符前面最近的两个操作数执行对应运算，合体为一个操作数
      - 机算
        - ①从左往右扫描下一个元素，直到处理完所有元素
        - ②若扫描到操作数则压入栈，并回到①；否则执行③
        - ③若扫描到运算符，则弹出两个栈顶元素，执行相应运算，运算结果压回栈顶，回到①
    - 中缀表达式转前缀表达式（手算）
      - ① 确定中缀表达式中各个运算符的运算顺序
      - ② 选择下一个运算符，按照「运算符 左操作数 右操作数」的方式组合成一个新的操作数
      - ③ 如果还有运算符没被处理，就继续 ②
    - 前缀表达式的计算
      - ①从右往左扫描下一个元素，直到处理完所有元素
      - ②若扫描到操作数则压入栈，并回到①；否则执行③
      - ③若扫描到运算符，则弹出两个栈顶元素，执行相应运算，运算结果压回栈顶，回到①
    - 中缀表达式转后缀表达式
      - 手算
        - ① 确定中缀表达式中各个运算符的运算顺序
        - ② 选择下一个运算符，按照「左操作数 右操作数 运算符」的方式组合成一个新的操作数
        - ③ 如果还有运算符没被处理，就继续 ②
        - ”左优先”原则：只要左边的运算符能先计算，就优先算左边的
      - 机算：从左到右处理各个元素，直到末尾。可能遇到三种情况：
        - ① 遇到操作数。直接加入后缀表达式。
        - ② 遇到界限符。遇到“(”直接入栈；遇到“)”则依次弹出栈内运算符并加入后缀表达式，直到弹出“(”为止。注意：“(”不加入后缀表达式。
        - ③ 遇到运算符。依次弹出栈中优先级高于或等于当前运算符的所有运算符，并加入后缀表达式，若碰到“(”或栈空则停止。之后再把当前运算符入栈。
    - 中缀表达式的计算（用栈实现）
      - 初始化两个栈，操作数栈和运算符栈
      - 若扫描到操作数，压入操作数栈
      - 若扫描到运算符或界限符，则按照“中缀转后缀”相同的逻辑压入运算符栈（期间也会弹出运算符，每当弹出一个运算符时，就需要再弹出两个操作数栈的栈顶元素并执行相应运算，运算结果再压回操作数栈）
  - 递归
    - 计算正整数的阶乘 n!
      - 递归调用时，函数调用栈可称为“递归工作栈”
      - 每进入一层递归，就将递归调用所需信息压入栈顶
      - 每退出一层递归，就从栈顶弹出相应信息
      - ***缺点：太多层递归可能会导致栈溢出***
    - 求斐波那契数列
- **队列在计算机系统中的应用**
  - 解决主机与外部设备之间速度不匹配的问题　　利用队列先进先出的性质，实现对打印机与主机速度不匹配的协调功能
  - 解决由多用户引起的资源竞争问题　　将多个用户排成一个队列，然后利用先进先出的性质分别将CPU分配给不同的用户使用
  - 图的广度优先遍历

3.3~4矩阵的压缩存储（下）

特殊矩阵的压缩存储

数组的定义

概述
- 数组是由n个相同类型的数据元素构成的有限序列
- 每个数据元素称为一个数组元素
- 每个元素在n个线性关系中的序号称为该元素的下标,下标的取值范围称为数组的维界　下标从0开始计数

数组与线性表的关系
- 数组是线性表的推广
- 一维数组可视为一个线性表；二维数组可视为 其元素也是定长线性表的线性表

特点　数组一旦被定义，其维数和维界就不再改变

数组的存储结构

一个数组的所有元素在内存中占用一段连续的存储空间

多维数组　映射方法

按行优先
- 按照一行一行的存储
- 二维数组公式 $LOC(a_{i,j}) = LOC(a_{0,0}) + [i \times (h_2 + 1) + j] \times L$

按列优先
- 按照一列一列的存储
- 二维数组公式 $LOC(a_{i,j}) = LOC(a_{0,0}) + [j \times (h_1 + 1) + i] \times L$

考试的时候可以根据具体问题现场推演

矩阵的压缩存储

概述

压缩存储
- 指为多个值相同的元素只分配一个存储空间，对零元素不分配存储空间
- 目的是为了节省存储空间

特殊矩阵
- 指具有许多相同矩阵元素或零元素，并且这些相同矩阵元素或零元素的分布有一 定规律性的矩阵

常见的特殊矩阵
- 对称矩阵
- 上（下）三角矩阵
- 对角矩阵

特殊矩阵的压缩存储方法：找出特殊矩阵中值相同的矩阵元素的分布规律，把那些呈现规律 性分布的、值相同的多个矩阵元素压缩存储到一个存储空间中

特殊矩阵

对称矩阵

概念
- 关于左对角线对称的矩阵
- 上三角区：位于对角线上方的区域
- 下三角区：位于对角线下方的区域

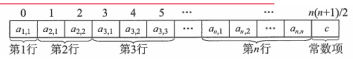
计算公式

$$k = \begin{cases} \frac{i(i-1)}{2} + j - 1, & i \geq j \text{（下三角区和主对角线元素）} \\ \frac{j(j-1)}{2} + i - 1, & i < j \text{（上三角区元素} a_{ij} = a_{ji} \text{）} \end{cases}$$

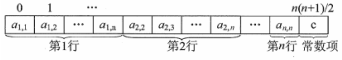
记不住公式的话记住推导过程，考试的时候可以直接带入试试

三角矩阵

下三角矩阵
- 上三角区的所有元素均为同一常量
- 存储完下三角区和主对角线上的元素之后,紧接着存储对角线上方的常量一次
- 元素下标之间的对应关系 $k = \begin{cases} \frac{i(i-1)}{2} + j - 1, & i \geq j \text{（下三角区和主对角线元素）} \\ \frac{n(n+1)}{2}, & i < j \text{（上三角区元素）} \end{cases}$
- 下三角矩阵在内存中的压缩存储形式



上三角矩阵
- 下三角区的所有元素均为同一常量
- 只需存储主对角线、上三角区上的元素和下三角区的常量一次
- 元素下标之间的对应关系 $k = \begin{cases} \frac{(i-1)(2n-i+2)}{2} + (j-i), & i \leq j \text{（上三角区和主对角线元素）} \\ \frac{n(n+1)}{2}, & i > j \text{（下三角区元素）} \end{cases}$
- 上三角矩阵在内存中的压缩存储形式



三对角矩阵
- 又称为带状矩阵，以角线为中心的3条对角线的区域，其他区域的元素都为零
- 三对角矩阵A也可以采用压缩存储,将3条对角线上的元素按行优先方式存放在一维数组B中,且a,存放于B[0]中
- 在一维数组B中存放的下标为k=2i＋j-3

稀疏矩阵
- 矩阵中非零元素的个数t,相对矩阵元素的个数s来说非常少,即s>>t的矩阵称为稀疏矩阵
- 存储方式　三元组　将非零元素及其相应的行和列构成一个三元组(行标，列标，值)
- 注意：稀疏矩阵压缩存储后便失去了随机存取特性
- 稀疏矩阵的三元组既可以采用数组存储,也可以采用十字链表法存储